МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
Кафедра информатики и компьютерных систем

САКОВИЧ
Иван Анатольевич

РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ ДЛЯ ПОСТРОЕНИЯ ЛОГИСТИЧЕСКОЙ КАРТЫ МАРШРУТОВ КОММЕРЧЕСКОГО ТРАНСПОРТА

Дипломная работа

Научный руководитель:
Кандидат технических наук,
Доцент
И.П. Стецко

Допущен к защите
«____»_____ 2015г.
Зав. кафедрой информатики и компьютерных систем
доктор технических наук, профессор С.Г. Мулярчик
_____

Минск, 2015

СОДЕРЖАНИЕ

Реферат

Дипломная работа содержит 55 страниц, 12 рисунков, библиография содержит 9 наименований.

Ключевые слова: ТРАНСПОРТНАЯ ЗАДАЧА, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, РАССТОЯНИЕ МЕЖДУ ГЕОГРАФИЧЕСКИМИ КООРДИНАТАМИ

Целью дипломной работы является разработка программного модуля для построения логистической карты маршрутов автотранспорта производственного предприятия на примере коммунального унитарного предприятия (КУП) «Минскхлебпром».

В результате выполнения настоящей дипломной работы была изучена специфика работы логистического отдела промышленного предприятия на примере КУП «Минскхлебпром» и формализовано условие транспортной задачи из многочисленных требований предприятия.

В качестве решения данной проблемы было рассмотрены несколько универсальных программных пакетов для решения транспортной задачи. Поскольку ни одно из них не удовлетворяет в полной мере составленному условию, было решено разработать собственный программный модуль для решения поставленной задачи.

Для этого было исследовано несколько алгоритмов для решения транспортной задачи и выбран алгоритм, наиболее полно отвечающий требованиям предприятия. В результате был реализован алгоритм, который в состоянии рассчитать приемлемую логистическую карту маршрутов, учитывая все условия поставленной транспортной задачи.

Рэферат

Дыпломная праца змяшчае 55 старонак, 12 малюнкаў, бібліяграфія змяшчае 9 найменняў.

Ключавыя словы: ТРАНСПАРТНЫЯ ЗАДАЧЫ, ГЕНЕТЫЧНЫ АЛГАРЫТМ, АДЛЕГЛАСЦЬ ПАМІЖ ГЕАГРАФІЧНЫМІ КААРДЫНАТАМІ

Мэтай дыпломнай працы з'яўляецца распрацоўка праграмнага модуля для пабудовы лагістычнай карты маршрутаў аўтатранспарту вытворчага прадпрыемства на прыкладзе камунальнага ўнітарнага прадпрыемства (КУП) «Мінскхлебпрам».

У выніку выканання сапраўднай дыпломнай працы была вывучана спецыфіка працы лагістычнага аддзела прамысловага прадпрыемства на прыкладзе КУП «Мінскхлебпрам» і фармалізавана ўмова транспартнай задачы з шматлікіх патрабаванняў прадпрыемства.

У якасці рашэння дадзенай праблемы было разгледжаны некалькі універсальных праграмных пакетаў для вырашэння транспартнай задачы. Паколькі ні адно з іх не задавальняе ў поўнай меры складзеным умове, было вырашана распрацаваць уласны праграмны модуль для вырашэння пастаўленай задачы.

Для гэтага было даследавана некалькі алгарытмаў для вырашэння транспартнай задачы і абраны алгарытм, найбольш поўна адказвае патрабаванням прадпрыемства. У выніку быў рэалізаваны алгарытм, які ў стане разлічыць прымальную лагістычную карту маршрутаў, улічваючы ўсе ўмовы пастаўленай транспартнай задачы.

Abstract

Thesis contains 55 pages, 12 figures, bibliography contains 9 references.

Keywords: VEHICLE ROUTING PROBLEM, GENETIC ALGORITHMS, DISTANCE BETWEEN GEOGRAPHIC COORDINATES

The aim of the thesis is to develop a software module for the construction of the logistics route maps, vehicle manufacturing plant on the example of communal unitary enterprise (CUE) "Minskhlebprom."

As a result of this thesis there were studied the specifics of the work logistics department of an industrial enterprise on the example of CUE "Minskhlebprom" and formalized condition of the vehicle routing problem of the numerous requirements of the enterprise.

Several universal software package for solving the vehicle routing problem has been overviewed as a solution to this problem. Since none of them does not fully satisfy compiled condition, it was decided to develop its own software module for this task.

There were studied several algorithms for solving the vehicle routing problem and the algorithm is selected that best meets the requirements of the enterprise. As a result, the algorithm that is able to calculate the acceptable logistical route map, taking into account all the conditions set by the transportation problem.

Введение

В настоящее время для большинства производственных предприятий Республики Беларусь весьма актуальной является задача сокращения транспортных издержек. Особую остроту эта проблема приобрела для коммунального унитарного предприятия (КУП) «Минскхлебпром», жизненно важную разнообразную и массовую продукцию которого требуется ежедневно по графику развозить по нескольким тысячам торговых объектов Минска и республики. Для чего у предприятия имеется большой парк собственного автотранспорта, а также активно привлекаются сторонние компании, осуществляющие коммерческие грузоперевозки. Возникающие при этом специфические логистические задачи тяжело решать даже опытным диспетчерам. Поиск же эффективных решений, позволяющих не просто обеспечить развозку продукции, но и по возможности сократить транспортные издержки, требует применения современных компьютерных средств – специального программного обеспечения, позволяющего оперативно строить логистические карты маршрутов грузовых автомобилей.

Целью настоящей работы является разработка программного модуля для построения логистической карты маршрутов автотранспорта производственного предприятия на примере КУП «Минскхлебпром».

Себестоимость перевозок складывается из постоянных и переменных затрат [1]. К постоянным затратам относятся заработная плата и социальное обеспечение водителей, диспетчеров, специалистов по ремонту и обслуживанию транспорта и руководителей. Основная экономия достигается за счет снижения переменных затрат, состоящих из стоимости топлива, горюче-смазочных материалов, шин, запасных частей, плановых технических обслуживаний, амортизации автомобилей. Оплата услуг перевозки продукции сторонними компаниями также относится к переменным затратам.

Собственный автопарк КУП «Минскхлебпром» насчитывает около двухсот единиц автотранспорта, из них более ста единиц оборудованы

системой GPS-мониторинга перемещений. Есть шесть автостоянок, которые расположены рядом с соответствующими хлебозаводами, на каждой из которых базируются порядка 15 автомобилей. Автомобили, в свою очередь, различаются емкостью и предназначением, бывают автомобили для перевозки тортов и для перевозки остальной продукции.

С точки зрения логистики продукцию КУП «Минскхлебпром» можно разделить на более чем 200 типов. Помимо очевидного различия между хлебом, хлебобулочными изделиями, сухарно-бараночной продукции, печеньем и тортами, продукция, выпускаемая предприятием Минскхлебпром, различается также упаковкой, но которой может быть нанесен логотип магазина, в котором эта продукция должна реализовываться.

Необходимо развести продукцию кратчайшим путем, удовлетворив требования заказчиков к типу изделия, а также попав в установленное временное окно. Хлеб, хлебобулочных изделия и торты необходимо привозить в установленное временное окно, в то время как доставка сухарно-бараночных изделий и печенья не требует точного времени прибытия.

Время на поиск эффективного маршрута развоза продукции является приемлемым, если оно не превышает 15 минут, так как список заказов с указанием времени прибытия утверждается незадолго до начала загрузки продукции в грузовые автомобили. Возможность изменять маршрут в процессе развоза товара не требуется.

1. ПОСТАНОВКА ЗАДАЧИ

Для решения транспортной задачи необходимо определить список всех возможных требований транспортной задачи, а также указать, какие из них нам необходимы, а какие не требуются. Все требования можно классифицировать по трем группам:

– требования к грузовым автомобилям;

– требования к объектам, в которые следует осуществить доставку;

– требования к поиску логистической карты маршрутов.

1.1 Требования к грузовым автомобилям

Координаты начала движения автомобиля. Это очевидный основной параметр любой транспортной задачи, который безусловно нужно учитывать.

Поддержка нескольких начальных позиций для разных автомобилей. В состав КУП «Минскхлебпром» входят шесть хлебозаводов, на каждом из которых есть стоянка со своим автопарком. Поэтому необходимо учитывать то, что разные автомобили начинают движение из разных мест.

Окончание маршрута грузовика в его начальной позиции. Поскольку все грузовики, принадлежащие предприятию, прикреплены к соответствующей стоянке, а на расценки арендного транспорта этот нюанс не влияет, то любой маршрут должен оканчиваться в том же месте, где и начался.

Тип грузового автомобиля. Существуют специальные автомобили, предусмотренные для перевозки тортов, а все остальные автомобили могут перевозить любую остальную продукцию. Следовательно, требуется принять в расчет два типа автомобилей.

Емкость грузового отсека и грузоподъемность. Превысить порог грузоподъемности грузовика практически невозможно, поскольку плотность продукции КУП «Минскхлебпром» достаточно низкая. Из этого следует, что нужно учитывать только емкость грузового отсека.

Время работы грузового автомобиля. Точное время обеденного перерыва водителей грузовиков не определено, а окончание рабочего дня целесообразнее контролировать через временное окно. Окончание временного окна любого магазина должно быть не позже окончания рабочего дня водителей. Так что принимать в расчет время работы не требуется.

Стоимость работы автомобиля за единицу времени и за единицу пройденного расстояния. Водители на КУП «Минскхлебпром» работают на окладе, то есть эта статья расходов предприятия не зависит от того, едет водитель или ожидает (простаивает), учитывать стоимость работы за единицу времени не требуется. А вот стоимость работы за единицу пройденного расстояния необходимо принять во внимание, так как стоимость топлива, горюче-смазочных материалов, шин, запасных частей, плановых технических обслуживаний, амортизации автомобилей зависит прямо пропорционально от пройденного расстояния [1].

1.2 Требования к объектам, к которым необходимо осуществить доставку

Местоположение заведения заказчика. Это основной параметр любой транспортной задачи, так что очевидно – его нужно учитывать.

Необходимый объем товара. Поскольку заказчики «Минскхлебпрома» варьируются от маленьких столовых или павильонов на мини-рынках до супермаркетов, то и необходимый объем товара им нужен различный, в том числе иногда даже больший, чем может привезти один грузовой автомобиль. Это важный параметр, который алгоритм нахождения логистической карты обязательно должен учитывать.

Временное окно, в которое необходимо обеспечить доставку продукции. В контракте об осуществлении доставки указан временной диапазон, но не для всех типов товара, а только для хлеба, хлебобулочных изделий и тортов. Для остальной продукции временное окно должно равняться времени работы погрузочного участка.

Время обслуживания автомобиля грузчиками. Заказчик обязан обеспечить разгрузку автомобиля в течение указанного в контракте времени [1]. Решение транспортной задачи должно учитывать это время в поиске карты маршрутов.

1.3 Требования к поиску логистической карты маршрутов

Критерий оптимизации логистической карты маршрутов. Оптимизировать маршруты можно по времени и по расстоянию. Для КУП Минскхлебпром необходимо оптимизировать карту маршрутов с целью уменьшения суммы длин всех ее маршрутов, при этом удовлетворив все требования заказчика.

Время на поиск оптимального маршрута. Скорость завершения поиска очень важна, так как список заказов с указанием времени прибытия утверждается незадолго до начала загрузки продукции в грузовые автомобили. Было решено что 15 минут являются приемлемым временем для решения транспортной задачи.

Возможность изменять маршрут в процессе развоза товара. Предприятие «Минскхлебпром» не нуждается в такой возможности, так как все автомобили загружаются необходимой продукцией в начале рабочего дня согласно составленной транспортной накладной и в дальнейшем изменять их маршрут не имеет смысла.

Необходимость рассчитать карту маршрутов с несколькими ходками для некоторых грузовиков. Поскольку суммарный объем заказанной продукции на один день может превышать суммарную емкость всего доступного автотранспорта предприятия, то некоторым водителям нужно будет вернуться для еще одной загрузки в течение рабочего дня.

Учитывание топологии улиц в поиске карты маршрутов. Если топология улиц не учитывается, то вероятность нахождения оптимального маршрута уменьшится. Желательно принять в расчет топологию дорог в той или иной степени.

2. ОБЗОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ТРАНСПОРТНОЙ ЗАДАЧИ

2.1 Обзор программного обеспечения для решения транспортной задачи

Задача сокращения транспортных издержек актуальна во всем мире. Крупнейшие мировые компании, нуждающиеся в средствах для решения транспортной задачи, используют проприетарное программное обеспечение, специально разработанное для них. Тем не менее на рынке существует огромное число различных готовых веб и клиентских приложений для решения транспортной проблемы. Было рассмотрено порядка десяти распространенных сред для решения логистической задачи, и выбраны две наиболее подходящие под требования предприятия «Минскхлебпром» для дальнейшего изучения, а именно:

– Open Door Logistics Studio;

– Heuristic Lab.

Программа *Open Door Logistics Studio* является наиболее популярным программным обеспечением, распространяющимся бесплатно для некоммерческого использования. Реализованный алгоритм позволяет найти решение, учитывая емкость грузового отсека автомобилей и необходимый объем продукции, заказанной заказчиком, временные окна, продолжительность разгрузки. Также эта программа позволяет найти логистическую карту маршрутов, учитывая топологию дорог, при этом все равно обеспечивает высокую скорость поиска маршрута.

Тем не менее приложение не позволяет рассчитать маршрут с несколькими ходками. В случае превышения объема заказанной продукции над суммарным объемом всех грузовых автомобилей, программа не строит маршруты к некоторым заказчикам и оставляет их не обслуженными. Из

вышесказанного можно сделать вывод, что программа ODLS не применима для планирования логистики в КУП «Минскхлебпром».

Программа *Heuristic Lab* является универсальной, предоставляет различные алгоритмы для решения большого числа NP-сложных задач, в том числе транспортной задачи. Из всех разновидностей транспортных задач, которые возможно решить с помощью Heuristic Lab, наиболее подходящая требованиям КУП «Минскхлебпром» является транспортная проблема с учетом объема кузовов, временных окон, необходимого объема товара заказчикам и позволяет найти карту маршрутов с несколькими ходками грузовых автомобилей. К сожалению, решение находится только при условии одного депо, в котором загружаются и начинают движение автомобили. В результате это программное обеспечение тоже не подойдет требованиям предприятия «Минскхлебпром».

2.2 Классификация и сравнение алгоритмов поиска логистической карты маршрутов

Впервые решением транспортной задачи в научном плане занялись в начале 1940-х годов, когда учеными Л. Канторовичем и М. Гавуриным был разработан метод потенциалов для решения транспортной задачи. Позднее вычислительный алгоритм был доработан Дж. Данцигом и с развитием вычислительной техники метод потенциалов все еще является актуальным [7]. С тех пор было разработано или адаптировано множество алгоритмов решения транспортной задачи, среди которых стоит выделить генетический алгоритм.

2.2.1 Постановка транспортной задачи для решения методом потенциалов

Имеется m поставщиков $A_1, A_2, \dots, A_m$, у которых сосредоточены запасы одного и того же груза в количестве $a_1, a_2, \dots, a_m$ единиц, соответственно. Этот груз нужно доставить n потребителям $B_1, B_2, \dots, B_n$, заказавшим $b_1, b_2, \dots, b_n$ единиц этого груза, соответственно. Известны также все расстояния C_{ij} от

поставщика A_i к потребителю B_j. Требуется составить такой план перевозок, при котором общая длина маршрутов всех перевозок была бы минимальной.

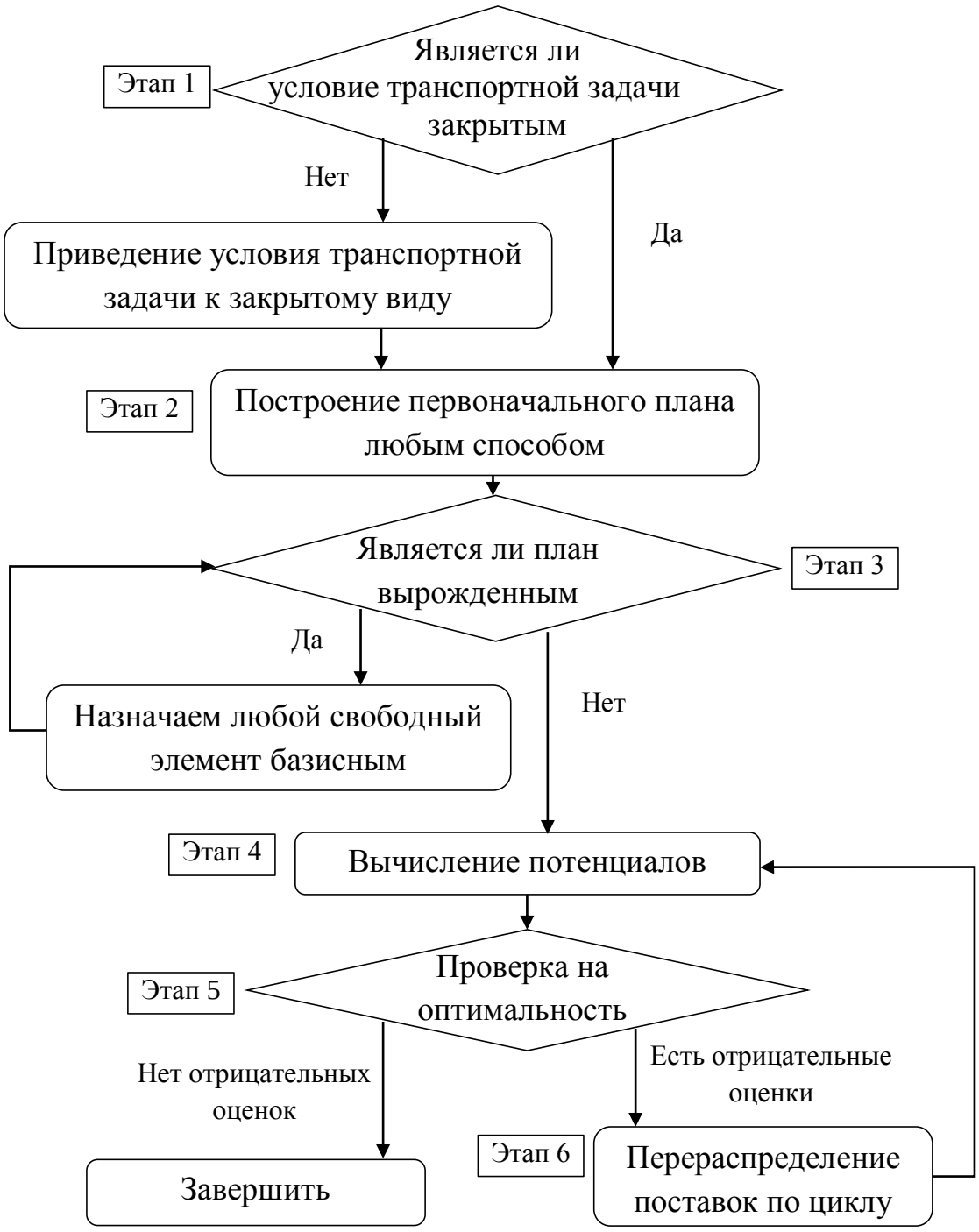

Рисунок 2.1 – блок-схема метода потенциалов

Обозначим суммарный запас груза у всех поставщиков символом a, суммарную потребность в грузе у всех потребителей – символом b. Транспортная задача называется закрытой, если $a = b$. Если же $a \neq b$, то транспортная задача называется открытой. В случае открытой задачи при $a <$

b весь груз будет вывезен, однако будут недопоставки груза наиболее далеким потребителям. При $a > b$, наоборот, будут удовлетворены все потребители, но часть груза останется на складах наиболее далеких поставщиков.

В заключение постановки задачи необходимо ввести понятия *количество груза* и *план перевозок*. Пусть x_{ij} – количество груза, отправляемого поставщиком A_i потребителю B_j, где $x_{ij} \geq 0$. Матрица X с неотрицательными элементами x_{ij} называется планом перевозок.

Рассмотрим подробнее этапы алгоритма, приведенного на рисунке 2.1.

Этап 1 – Проверка условия транспортной задачи на закрытость.

Решение транспортной задачи начинается с выяснения вопроса о том, является ли задача открытой или закрытой. Если задача является открытой, то необходимо провести процедуру закрытия задачи. С этой целью при $a < b$ добавляем фиктивного поставщика A'_{m+1} с запасом груза $a'_{m+1} = b - a$. Если же $a > b$, то добавляем фиктивного потребителя B'_{n+1} с заказом груза $b'_{n+1} = a - b$. В обоих случаях соответствующие фиктивным объектам расстояния перевозок C'_{ij} полагаем равными нулю. В результате сумма длин всех маршрутов не изменяется.

Этап 2 – Составление первоначального плана перевозок.

Транспортная задача относится к задачам линейного программирования, и ее можно было бы решить симплекс-методом [7]. Но поскольку система ограничений транспортной задачи проще, чем система ограничений основной задачи линейного программирования, то это дает возможность вместо использования объемных симплекс-таблиц применить более удобный метод, который состоит из следующих этапов:

– Составление первоначального плана перевозок;

– Последовательные улучшения плана перевозок (перераспределение поставок) до тех пор, пока план перевозок не станет оптимальным.

Решение транспортной задачи начинается с поиска допустимого начального решения (плана перевозок), чтобы все запасы поставщиков были распределены по потребителям. Допустимое начальное решение не обязательно оказывается оптимальным, а метод его нахождения может быть как простейшим (метод северо-западного угла или аналоги) или более сложным и приближенным к оптимальному решению (метод минимального расстояния, метод Фогеля), или же вообще произвольным.

Рассмотрим подробнее составление начального плана методом минимальных расстояний. Построение плана начнем с элемента матрицы с наименьшим расстоянием C_{ij}. При наличии нескольких элементов с одинаковыми расстояниями выберем любую из них. Пусть это будет элемент (i, j). Создадим матрицу и назовем ее базисной. Запишем на позицию (i, j) в базисной матрице значение $x_{ij} = min\ (a_i, b_j)$. Если $a_i < b_j$, то запасы поставщика A_i исчерпаны, а потребителю B_j требуется еще $b'_j = b_j - a_i$ единиц груза. Поэтому, не принимая более во внимание i-ю строку, снова ищем клетку с наименьшей стоимостью перевозок и заполняем ее с учетом изменившихся потребностей. В случае $a_i > b_j$ из рассмотрения исключается j-й столбец, а запасы A_i полагаются равными $a'_i = a_i - b_j$. Продолжаем этот процесс до тех пор, пока все запасы не будут исчерпаны, а все потребности удовлетворены.

Необходимо отметить, что при наличии в таблице клеток с одинаковыми тарифами, планы, полученные с помощью этого метода, могут быть разными, однако они, несомненно, ближе к оптимальному плану, чем план, составленный по методу северо-западного угла.

Этап 3 – Проверка на вырожденность первоначального плана

Прежде, чем перейти к анализу оптимальности планов и способам их улучшения, выясним, каким требованиям должны удовлетворять составляемые планы. В соответствии с определением плана перевозок у матрицы $X = \{x_{ij}\}$ сумма элементов i-й строки равняется a_i, $i = 1, 2, \ldots, m$, а сумма элементов j-о столбца равняется b_j, $j = 1, 2, \ldots, n$. Условие закрытости

транспортной задачи $a = b$ означает, что среди $m + n$ уравнений системы независимыми являются только $m + n - 1$ уравнений, поэтому в любом базисном решении этой системы должно быть $m + n - 1$ базисных переменных. Поскольку свободные переменные в таком решении равны нулю, то в базисной матрице им будут соответствовать элементы равные «-1».

Элементы матрицы, в которые записаны неотрицательные перевозки, называются *базисными*, а элементы равные «-1» – *свободными*.

План называется *вырожденным*, если количество базисных элементов в нем меньше, чем $m + n - 1$.

Если на каком-то этапе решения получился вырожденный план, то его необходимо пополнить, изменив элемент равный «-1» на нуль и превратив, тем самым, этот элемент в базисный. Общий баланс и суммарная дистанция перевозок плана при этом не изменятся [7]. Однако проводить пополнение плана, выбирая элемент произвольно, нельзя. Приведем условия, которым должен соответствовать пополненный план. Для этого введем понятия цикл и ацикличность:

Циклом в транспортной таблице называется несколько элементов базисной матрицы, соединенных замкнутой ломаной линией так, чтобы две соседние вершины ломаной были расположены либо в одной строке, либо в одном столбце. Ломаная линия может иметь точки самопересечения, но не в элементах цикла.

План называется *ациклическим*, если его базисная матрица не содержит циклов.

Оптимальные планы являются ациклическими, поэтому и первоначальный план также должен удовлетворять этому требованию. Заметим, что планы, полученные с помощью метода северо-западного угла или метода наименьшего расстояния, являются ациклическими. Однако если

план оказался вырожденным, то при его пополнении требование ацикличности необходимо учитывать.

Этап 4 – Вычисление потенциалов

Для анализа полученных планов и их последующего улучшения удобно ввести дополнительные характеристики пунктов отправления и назначения, называемые потенциалами.

Сопоставим каждому поставщику A_i и каждому потребителю B_j величины U_i и V_j соответственно так, чтобы для всех базисных элементов были выполнены соотношения (2.1):

$$U_i + V_j = C_{ij},\ \ i = 1, 2, \dots , m;\ \ j = 1, 2, \dots , n. \qquad (2.1)$$

Поскольку число базисных элементов в плане равно $m + n - 1$ (вырожденные планы должны быть предварительно пополнены), то для определения потенциалов получается система из $m + n - 1$ уравнений с $m + n$ неизвестными. Такая система имеет бесконечное множество решений. Нам требуется любое ее решение. Обычно для простоты полагают потенциал U_1 равным нулю и затем вычисляют остальные потенциалы.

Этап 5 – Проверка на оптимальность составленного плана

Для каждого свободного элемента базисной матрицы вычислим разности по формуле 2.2:

$$\Delta C_{ij} = C_{ij} - (U_i + V_j) \qquad (2.2)$$

И запишем полученные значения в матрицу оценок на позиции (i, j). Заметим, что для базисных элементов выполнено соотношение $\Delta C_{ij} = 0$, и этим фактом можно пользоваться для контроля правильности нахождения потенциалов.

План является оптимальным, если все разности $\Delta C_{ij} = 0$. В противном случае план можно улучшить следующим способом.

Этап 6 – Перераспределение поставок

Найдем элемент в матрице оценок с наибольшей по абсолютной величине отрицательной разностью ΔC_{ij} и начнем строить цикл в базисной матрице из его позиции (i, j). Остальные углы цикла должны попадать на позиции базисных элементов. Такой цикл всегда существует и единственен.

Заметим, что в новом плане суммы элементов по строкам и столбцам должны остаться прежними, поэтому изменение значения одного элемента базисной матрицы повлечет за собой соответствующие изменения значений всех остальных элементов этого цикла. Так как свободный элемент, на позиции которого начался цикл, будет увеличен, то условно присвоим знак «+». Теперь пройдем по всей ломаной цикла в любом направлении, присваивая поочередно знаки «+» и «-» соответствующим элементам.

Груз будет перераспределен по элементам базисной матрицы, соответствующих углам цикла на величину $\Delta x = min\ x_{ij}$ следующим образом. В клетках со знаком «+» значение перевозки нужно увеличить на величину Δx, а в клетках со знаком «-» – уменьшить на величину Δx. Так как после пересчета у нас добавилась лишний базисный элемент, то их количество необходимо сократить, изменив значение любого базисного элемента на -1, тем самым сделав его свободным.

После этого полученный план проверяется на оптимальность описанным выше способом. Перераспределение груза производится до тех пор, пока очередной план не станет оптимальным. Как только все ΔC_{ij} становятся неотрицательными, действие алгоритма завершается и оптимальная логистическая карта маршрутов найдена.

Хотя данный алгоритм был разработан для вычисления решения транспортной задачи на бумаге, он хорошо подходит и легко реализуется на вычислительной технике и дает решения за сравнительно короткое время. К сожалению он не позволяет учесть временные окна, в которые нужно попасть, доставляя товары.

2.2.2 Генетический алгоритм для решения транспортной проблемы

Генетические алгоритмы хорошо себя зарекомендовали как средство глобальной оптимизации, в том числе оптимизации транспортной задачи. Суть генетических алгоритмов заключается в моделировании процесса естественного отбора как средства нахождения оптимального решения, при этом используется механизм передачи наследственной информации из поколения в поколение. В качестве меры успешности организма используется целевая функция оптимизации, которая в терминах генетического алгоритма называется фитнес-функцией. В общем случае структура генетического алгоритма состоит из следующих операторов: инициализации, селекции, кроссинговера и мутации. Далее будут подробно описаны данные операторы, так как они являются основой представляемого генетического алгоритма.

Оператор инициализации должен построить N предварительных планов перевозки. Предварительные планы можно строить различными способами, начиная случайным образом и заканчивая методом на основе метода конструирования маршрутов Соломона. Преимущество метода Соломона в том, что он позволяет не только строить новые решения при инициализации популяции решений, но и достраивать решения (решения в которых часть клиентов обслужена, а часть еще нет) [6]. Основное требования к инициализации это что бы N предварительных планов значительно отличались друг от друга.

Приспособленность – качество составленного плана перевозок. Приспособленность можно вычесть например по формуле 2.3 [9]:

$$Z = S + u * k_u + b * k_b + a * k_a \qquad (2.3)$$

Где Z – оценка качества составленного плана, S – сумма длин всех маршрутов, u – незанятый объем в кузове, k_u – штраф за незанятый объем, b – время до закрытия временного окна, k_b – штраф за прибытие раньше временного окна, a – время опоздания, k_a – штраф за опоздание.

Значение штрафа за опоздание можно выбрать выше, чем значение штрафа за преждевременный приезд, а можно оставить равными. В случае перегруженности автомобиля приспособленность всего плана перевозок Z должна быть значительно выше, чем среднее значение Z остальных логистических карт маршрутов.

Оператор селекции – вычисление приспособленности для каждого плана перевозки в популяции и сортировка планов в порядке возрастания их приспособленности. Также удаление тех планов, индекс которых превышает выбранный N. Иногда в массиве планов оставляют только две трети от числа N, а оставшуюся одну треть N выбирают случайным образом из отсеянным, в том числе самых неприспособленных, ради разнообразия в последующих поколениях планов перевозки [8].

Оператор скрещивания. Для решения транспортной задачи лучше всего реализовать оператор скрещивания по аналогии с адаптивной памятью в методе поиска с запретами. Количество родителей, участвующих в скрещивании, определяется числом M, увеличение M позволяет более эффективно передавать свойства (маршруты) родителей потомкам, сходимость алгоритма увеличивается, но это грозит опасностью попадания в локальный минимум, далекий от глобального минимума. При уменьшении M растет разнообразие в популяции [6]. Эксперименты показали, что алгоритм дает наилучшие результаты при M от 2 до 4 [6]. Далее следует подробное описание этапов, выполняемых оператором скрешивания:

1. Оператор скрешивания выбирает число M – количество решений (особей) участвующих в скрещивания.

2. Выбирает M случайных планов перевозки из популяции N.

3. Маршруты из выбранных решений объединяет в одно решение. Данное решение назовем объединенным решением.

4. Пока в объединенном решении есть маршруты, рператор скрешивания делает следующее: выбирает случайным образом маршрут и переносит его в новое решение, при этом убирая его из объединенного решения; удаляет в объединенном решении все маршруты, в которых есть клиенты из выбранного решения;

5. Вставляет не обслуженных клиентов в новое решение случайным образом.

6. Добавляет получившийся план в основную популяцию.

Операторы мутации. В генетическом алгоритме можно применять несколько операторов мутации для решения транспортной проблемы. Разные операторы мутации и комбинации этих операторов применяются на разных этапах поиска. Основным для применения является оператор мутации 1. Он наименее затратный по вычислительным ресурсам. При применении данного оператора из решения удаляются один или несколько маршрутов для автомобиля целиком, затем клиенты с удаленных маршрутов снова вставляются в решение случайным образом [6].

Оператор мутации 2 применяется например на завершающих этапах поиска, когда много поколений подряд не улучшают приспособленность лучшего плана перевозок. При применении оператора мутации 2 из решения удаляются отдельные клиенты, и затем заново вставляются в маршрут. В результате применения операторов мутации мы получаем новое решение, которое расширяет область поиска за счет нового набора маршрутов.

На рисунке 2.2 изображена блок-схема генетического алгоритма.

Генетический алгоритм является очень гибким, варьируя множество параметров которого можно добиться достаточно быстрого решения транспортной задачи с любыми комбинациями требований.

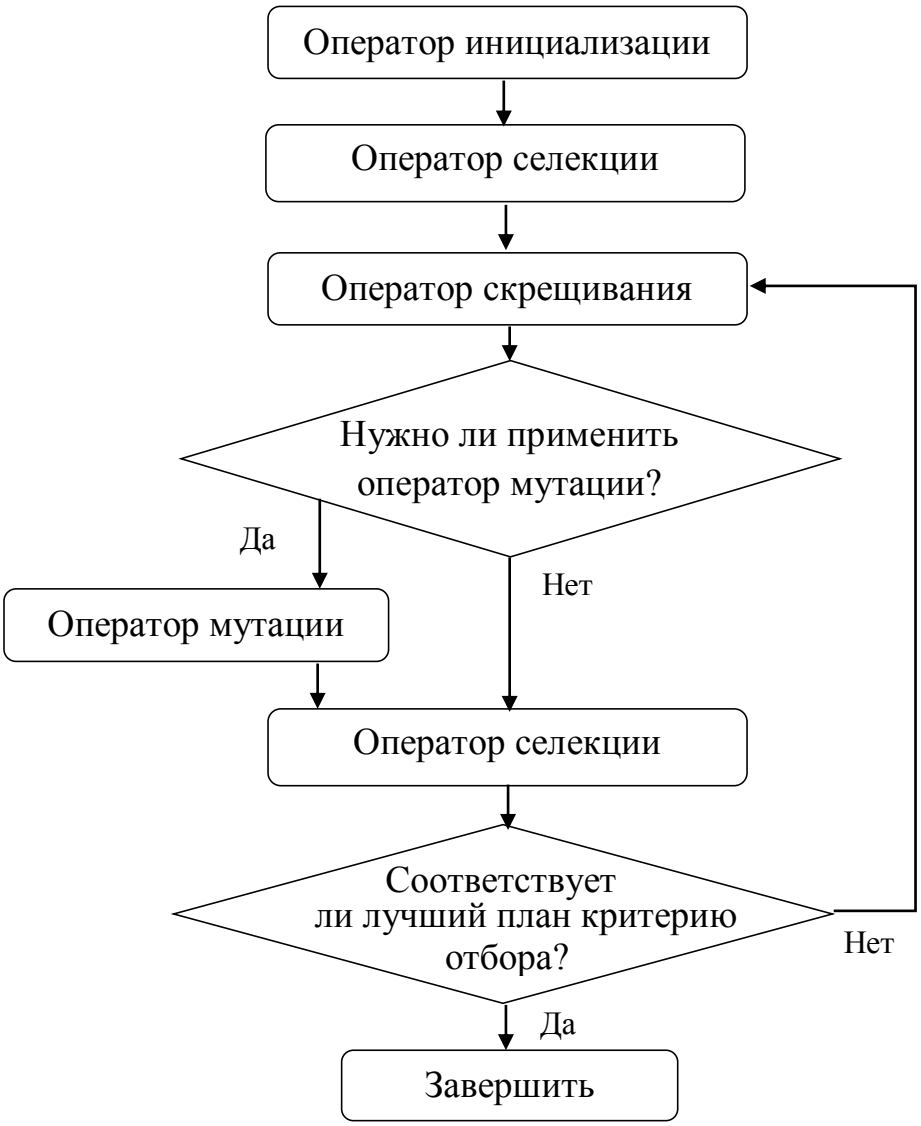

Рисунок 2.2 – блок-схема генетического алгоритма

2.3 Выбор приемлемого алгоритма для практической реализации

Адаптация генетического алгоритма для решения транспортной задачи является наиболее подходящей по ряду причин:

– Сравнительно простая реализация.

– Возможность решить задачу с любым условием.

– Высокое быстродействие.

Генетический алгоритм состоит из четырех основных операторов, из которых самый сложный и ресурсоемкий оператор вычисления приспособленности каждого плана в популяции решений. В частности

необходимо уделить особое внимание методу расчета приспособленности, так что бы приспособленность зависела от пройденного расстояния, просроченного времени до открытия и после закрытия временного окна, нормированного по пройденному расстоянию, и от недогруженности грузового автомобиля, также нормированного по расстоянию.

Этот алгоритм позволяет решить задачу с любым условием, в том числе транспортную задачу с такими сложными условиями, как поиск логистической карты маршрутов с возможностью вернуться автомобилям в соответствующий хлебозавод и загрузиться продукцией еще раз или учет временных окон, в которые нужно попасть доставляя товар заказчику.

Поскольку генетический алгоритм работает не с одним решением, а с 300 – 500 решениями одновременно, то лучшее решение должно улучшаться каждое поколение, так как всегда найдутся решения (планы перевозок) в популяции, которые не попадут в локальный минимум, а будут стремиться к глобальному минимуму [6]. Благодаря этому обеспечивается сравнительно быстрая оптимизация логистической карты маршрутов.

3. РЕАЛИЗАЦИЯ ВЫБРАННОГО АЛГОРИТМА

Возможность быстро найти кратчайший путь между двумя точками крайне важна для любого алгоритма решения транспортной задачи. Поскольку быстрое определение пути между двумя точками учитывая топологию дорог само по себе крайне сложная задача, то вначале рассмотрим вычисление кратчайшего расстояния между двумя точками на сфере, а также рассмотрим API Google Maps для определения расстояния между двумя географическими координатами.

3.1 Вычисление кратчайшего расстояния между двумя географическими координатами

Форма Земли отличается от шара и имеет несколько сплющенную форму, близкую к сфероиду, но истинная фигура Земли отличается и от сфероида. Поверхность воды в морях и океанах в спокойном состоянии и условно продолженная под материками образует геоид [3]. Но поскольку КУП «Минскхлебпром» необходимо строить логистическую карту маршрутов только на территории Республики Беларусь, то для ускорения определения расстояния Землю примем за сферу.

Через любые две точки на поверхности сферы, если они не прямо противоположны друг другу (то есть не являются антиподами), можно провести уникальный большой круг, центр которого совпадает с центром сферы [2]. Две точки, разделяют большой круг на две дуги. Длина короткой дуги – кратчайшее расстояние между двумя точками и называется ортодромией. Между двумя точками-антиподами можно провести бесконечное количество больших кругов, но расстояние между ними будет одинаково на любом круге и равно половине окружности круга.

Вычислить кратчайшее расстояние можно с помощью сферической теоремы косинусов по формуле 3.1:

$$\Delta\sigma = arccos\ \{\ sin\ \varphi_1\ sin\ \varphi_2 + cos\ \varphi_1\ cos\ \varphi_2\ cos\ \Delta\lambda\} \quad (3.1)$$

Где $\varphi_1, \lambda_1;\ \varphi_2, \lambda_2$ – широта и долгота двух точек соответственно, $\Delta\lambda$ – разница координат по долготе, $\Delta\sigma$ – угловая разница.

В случае маленьких расстояний и небольшой разрядности вычисления, использование сферической теоремы косинусов может приводить к значительным ошибкам связанным с округлением [2]. Формула гаверсинусов (3.2) позволит этого избежать.

$$\Delta\sigma\ =\ 2\arcsin\left\{\sqrt{sin^2\ \left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos\varphi_1\cos\varphi_2\ \ sin^2\left(\frac{\Delta\lambda}{2}\right)}\right\} \quad (3.2)$$

Где $\varphi_1, \lambda_1;\ \varphi_2, \lambda_2$ – широта и долгота двух точек соответственно, $\Delta\lambda$ – разница координат по долготе, $\Delta\sigma$ – угловая разница.

Собственно расстояние вычисляется по формуле 3.3.

$$d = \Delta\sigma * R \quad (3.3)$$

Где d – расстояние между двумя точками, R – радиус Земли, $\Delta\sigma$ – угловая разница.

В результате вычислять расстояние между двумя точками будем по формуле гаверсинусов, а радиус земли примем равным 6 372 795 метров [2].

3.2 Средства Google Maps для определения расстояния между двумя точками

API «маршрутов» Google – это служба, которая рассчитывает маршруты между пунктами на карте с помощью HTTP-запроса [4]. Поддерживается поиск маршрутов для различных способов перемещения, в том числе на общественном транспорте, автомобиле, велосипеде или пешком. При поиске маршрутов пункты отправления, назначения, а также путевые точки могут указываться либо в виде текстовых запросов (например, "Minsk, Belarus"), либо в виде значений координат – широты и долготы.

На использование службы API «маршрутов» Google накладывается ограничение, составляющее 2500 запросов маршрутов в день. Далее необходимо будет получить больше 7500 маршрутов, на что потребуется четыре дня. Поиск автомобильного, велосипедного или пешеходного маршрута учитывается при расчете ежедневной квоты как один запрос. Поиск маршрута общественного транспорта учитывается как 4 запроса.

Запросы к службе маршрутов

Запрос API маршрутов имеет следующий вид:

```
http://maps.googleapis.com/maps/api/directions/output?parameters
```

здесь параметр `output` может принимать одно из следующих значений:

`json`, предписывающее выводить результаты в формате JavaScript Object Notation (JSON).

`xml`, предписывающее выводить результаты в формате XML.

Параметры запроса

Параметры разделяются амперсандами (&), что является стандартом для всех адресов URL. Ниже представлен список параметров, и перечислены их допустимые значения.

`origin` – адрес или текстовое значение широты и долготы отправного пункта маршрута. Если адрес передается в виде строки, для вычисления маршрута служба выполняет геокодирование строки и преобразует ее в значения координат широты и долготы. При передаче координат не допускаются пробелы между значениями широты и долготы.

`destination` – адрес или текстовое значение широты и долготы отправного пункта маршрута. Если адрес передается в виде строки, для вычисления маршрута служба выполняет геокодирование строки и преобразует ее в значения координат широты и долготы. При передаче координат не допускаются пробелы между значениями широты и долготы.

sensor указывает, исходит ли запрос маршрута от устройства с датчиком местоположения. Допустимые значения – true или false.

Элементы ответов службы маршрутов

Ответы службы маршрутов содержат два корневых элемента:

"status" содержит метаданные по запросу.

"routes" содержит массив маршрутов из исходной точки к пункту назначения.

Коды статуса

Поле "status", входящее в объект ответа службы маршрутов, содержит статус запроса и может включать в себя отладочную информацию для отслеживания сбоев службы. Поле "status" может содержать следующие значения:

OK указывает, что ответ содержит допустимый элемент result.

NOT_FOUND означает, что по крайней мере для одного указанного пункта (исходный, пункт назначения или путевая точка) не удалось выполнить геокодирование.

ZERO_RESULTS – указывает, что между исходной точкой и пунктом назначения не найдено ни одного маршрута.

MAX_WAYPOINTS_EXCEEDED означает, что в запросе указано слишком много элементов массива waypoints. Разрешено использовать массив waypoints, содержащий не более 8 элементов, плюс исходный и конечный пункты.

INVALID_REQUEST – указывает, что запрос является недопустимым.

OVER_QUERY_LIMIT – означает, что служба получила слишком много запросов от вашего приложения в разрешенный период времени.

REQUEST_DENIED указывает, что служба отклонила запрос вашего приложения.

UNKNOWN_ERROR означает, что обработка запроса маршрута невозможна из-за ошибки сервера. При повторной попытке запрос может быть успешно выполнен.

Маршруты

API маршрутов помещает результаты в массив routes (JSON). Даже если служба не возвращает никаких результатов (например, если исходная точка или пункт назначения не существуют), все равно возвращается пустой массив routes. (Число элементов в ответах службы в формате XML больше или равно нулю <route>.)

Каждый элемент массива routes содержит единственный результат для указанных исходного и конечного пунктов. В зависимости от числа указанных путевых точек маршрут может состоять из одного или нескольких элементов legs.

Каждый маршрут, входящий в поле routes, может содержать следующие поля:

summary включает краткое текстовое описание маршрута, подходящее для присвоения имени и устранения неоднозначности.

legs[] содержит массив с информацией об отрезке маршрута между двумя пунктами на заданном маршруте. Для каждой указанной путевой точки или пункта назначения будет представлен отдельный отрезок. (В массиве legs для маршрута без путевых точек будет содержаться только один отрезок.)

overview_polyline содержит объект с массивом закодированных элементов points, которые представляют приблизительный (сглаженный) путь результирующего маршрута.

Отрезки

Каждый элемент в массиве `legs` указывает на отдельный отрезок пути от исходного до конечного пункта в вычисленном маршруте. Маршруты без путевых точек состоят из одного "отрезка", а маршруты с одной или несколькими путевыми точками – из одного или нескольких отрезков, соответствующих конкретным участкам пути.

Все отрезки, входящие в поля `legs`, могут включать следующие поля:

`steps[]` содержит массив шагов с информацией о каждом шаге на отрезке пути.

`distance` обозначает общее расстояние, охватываемое этим отрезком, и представляет собой поле со следующими элементами:

`value` указывает расстояние в метрах.

`text` содержит удобочитаемое представление расстояния в единицах измерения, принятых в исходном пункте.

`duration` обозначает общую продолжительность прохождения этого отрезка в виде поля со следующими элементами:

`value` указывает продолжительность в секундах.

`text` содержит удобочитаемое представление продолжительности.

Если продолжительность неизвестна, то эти поля могут отсутствовать.

`start_location` содержит значения координат широты и долготы исходной точки этого отрезка. API маршрутов для вычисления маршрута между пунктами использует ближайший к начальному и конечному пунктам путь сообщения (обычно дорогу), поэтому исходный пункт `start_location` может отличаться от указанной исходной точки отрезка (например, если дорога находится далеко от исходной точки).

`end_location` содержит значения координат широты и долготы заданного пункта назначения этого отрезка. API маршрутов для вычисления

маршрута между пунктами использует ближайший к начальному и конечному пунктам путь сообщения (обычно дорогу), поэтому конечный пункт `end_location` может отличаться от указанного пункта назначения этого отрезка (например, если дорога находится далеко от пункта назначения).

`start_address` содержит удобочитаемый адрес (обычно улицу и номер дома), отражающий значение `start_location` для этого отрезка.

`end_address` содержит удобочитаемый адрес (обычно улицу и номер дома), отражающий значение `end_location` для этого отрезка.

Пример использующегося запроса для составления таблицы расстояний:

`http://maps.googleapis.com/maps/api/directions/json?origin=53.839696,27.475651&destination= 53.924543,27.576574&sensor=false`

Торговые объекты отбытия и назначения указаны в виде географических координат. Из ответа выделяем первый элемент массива `routes`, из него первый элемент массива `legs`, а из массива `legs` выбираем поле `distance`. Из этого поля необходимо взять переменную `value`, которая является искомым расстоянием между двумя географическими координатами с учетом топологии дорог.

3.3 Составление таблицы расстояний

На данный момент в городе Минск расположено 1454 объектов, которые являются клиентами предприятия «Минскхлебпром». При этом, для того, чтобы добраться от одного объекта к другому, расположенному по разные стороны дороги, порой нужно проехать до четырех километров. Или между двумя объектами, расположенными рядом по одну сторону дороги, в прямом направлении проехать легко, а в обратном нужно выполнять длинный объезд.

Поскольку в алгоритме расчета оптимальной логистической карты маршрутов необходимо использовать геометрической расстояние между объектами, то из за нюансов в организации дорожного движения могут возникать значительные ошибки.

Ради исключения подобных ошибок было решено составить таблицу, в которой будут содержаться расстояния по дорогам для пары объектов, кратчайшее расстояние между которыми меньше 400 метров. Именно для близлежащих объектов, разница между кратчайшим расстоянием и расстоянием, учитывающим топологию улиц, может быть наиболее существенной.

Таблица состоит из идентификатора исходного объекта, идентификатора объекта назначения и собственно расстояния между ними, учитывая топологию улиц. В таблицу расстояний попали 7868 записей, полученных от сервиса «маршрутов» Google Maps.

Граница в 400 метров была выбрана исходя из того, сколько запросов в день позволяет сделать сервис Google. Три дня на составление матрицы расстояний было принято приемлемым, следовательно нужно найти такое предельное эвклидово расстояние между объектами, что бы количество реальных расстояний по дорогам равнялось приблизительно 7500. Рисунок 3.1 показывает зависимость количества записей в таблице по оси Y от граничного расстояния между двумя объектами по оси X. На Рисунке 3.2 изображен укрупненный вид предыдущего графика в приемлемой области.

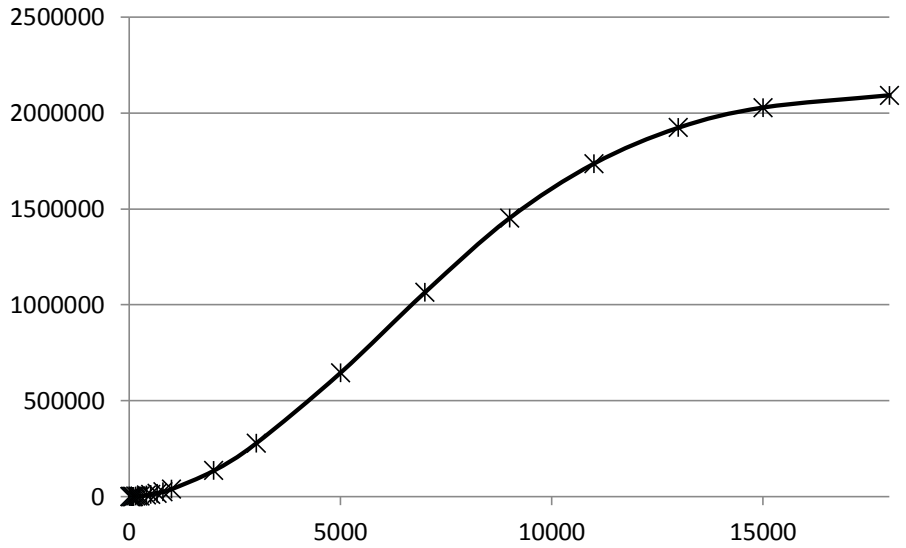

Рисунок 3.1 – зависимость количества записей в таблице от расстояния между двумя объектами

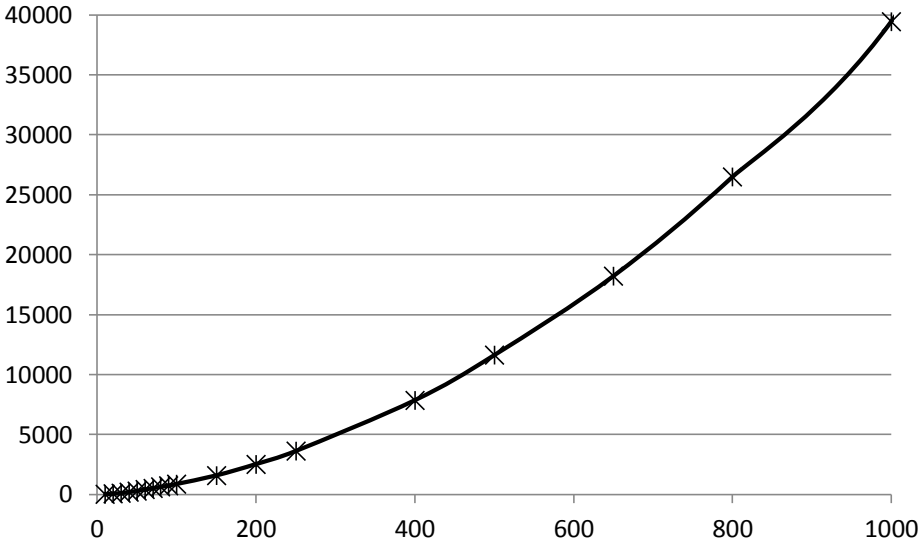

Рисунок 3.2 – зависимость количества записей в таблице от расстояния между двумя объектами (укрупненный вид)

Графики демонстрируют, что приемлемые 7868 расстояний для таблицы можно получить выбирая пары торговых объектов, кратчайшие расстояние между которыми меньше 400 метров.

3.4 Программная реализация генетического алгоритма для решения транспортной задачи

Суть генетических алгоритмов заключается в моделировании процесса естественного отбора как средства нахождения оптимального решения, при этом используется механизм передачи наследственной информации из поколения в поколение. В качестве особей популяции выступают предварительные планы перевозки, составленные случайным образом. Популяция представляет собой список классов `Plan`:

```
public static List<Plan> population = new List<Plan>();
```

Класс `Plan` в свою очередь включает в себя список классов `Car` и целочисленную переменную `fitness`, в которую будет заноситься значение приспособленности плана перевозки.

```
public class Plan
{
	public List<Car> car;
	public int fitness;
}
```

Класс Car состоит из списка классов Shop. В него будут включаться магазины, которые необходимо посетить данному автомобилю. Также в этом классе есть переменная типа `int`, которая равняется емкости кузова автомобиля, и переменная типа `PointLatLng`. `PointLatLng` является классом, который содержит в себе географические координаты хлебозавода, к которому прикреплен этот автомобиль.

```
public class Car
{
	public List<Shop> stops;
	public int maxCapacity;
	public PointLatLng basePos;
}
```

И наконец класс Shop. Он состоит из пяти целочисленных переменных: `ID`, `startTime`, `endTime`, `unloadingTime`, `needCapacity`, а также переменной класса `PointLatLng` с координатами магазина. `ID` является уникальным идентификатором, назначаемый предприятием «Минскхлебпром» каждому объекту, к которому необходимо доставлять продукцию. `startTime` и `endTime` это время начала и окончания временного окна соответственно, `unloadingTime` это время, за которое клиент обязуется разгрузить автомобиль. А `needCapacity` это объем заказанной продукции.

```
public class Shop
{
	public int ID;
	public int startTime;
	public int endTime;
	public int unloadingTime;
	public int needCapacity;
	public LatLng position;
}
```

Также необходимо объявить глобальную переменную `shops`, которая состоит из списка классов `Shop`, в которую будут занесены все магазины для посещения за день.

Далее заполняем переменную `population` десятью планами, планы заполняем списком автомобилей, готовых к работе, и в переменную `fitness` записываем «-1», так как в дальнейшем метод, вычисляющий приспособленность плана, будет рассчитывать приспособленность только для тех планов, переменная `fitness` в которых равняется «-1». Это позволит значительно сократить время нахождения оптимальной логистической карты маршрутов, так как в популяции некоторые планы могут оставаться долгое время без изменений. В каждый автомобиль добавляем магазины из списка `shops`, которые нужно посетить за день, таким образом, что бы каждый автомобиль остановился примерно одинаковое число раз, а также что бы каждый магазин был обслужен только одним автомобилем.

3.4.1 Реализация фитнес-функции

В качестве меры успешности организма используется целевая функция оптимизации, которая в терминах генетического алгоритма называется фитнес-функцией. В ее роли будет выступать метод названный `FitnessFunc`. Он состоит из цикла, который перебирает все планы в популяции для вычисления приспособленности каждого из них.

Первым делом в цикле проверяем, равняется ли приспособленность плана -1, если не равняется, то переходим к следующему плану в популяции.

```
if (population[p].fitness != -1)
     continue;
```

Далее объявляем три целочисленные переменные, `overallDistance`, `timePenalty` и `capacityPenalty`, которые будут участвовать в вычислении приспособленности. `overallDistance` означает дистанцию пройденную всеми автомобилями этого плана, `timePenalty` – суммарное время опоздания

или приезда раньше временного окна каждого автомобиля, capacityPenalty – сумма недогруженностей всех автомобилей.

Теперь необходимо перебрать все автомобили в плане и вычислить расстояние пройденное каждым грузовиком в отдельности. Сперва рассчитывается расстояние от соответствующего хлебозавода до первого магазина в списке stops. Если для данного автомобиля не запланированы остановки, то расстояние принимается равным нулю и цикл переходит к следующему автомобилю.

```
if (population[p].car[c].stops.Count == 0)
{
     carDistance = 0;
     continue;
}
else carDistance = Distance(population[p].car[c].basePos,
     population[p].car[c].stops[0].position);
```

Если остановки в магазинах для автомобиля предусмотрены, то далее будут объявлены две целочисленные переменные time и capacity. В переменную capacity записываем значение maxCapacity данного грузовика и она будет уменьшаться после посещения каждого объекта на величину равную needCapacity соответствующего магазина. Переменная time означает текущее время в минутах и равняется time = carDistance / 667; в качестве скорости взято значение 667 метров в минуту. В дальнейшем переменная time будет увеличиваться в зависимости от пройденного расстояния и значения unloadingTime каждого магазина.

Далее запускается цикл, который перебирает все магазины, запланированные для посещения автомобилем. В цикле первым делом вычисляется приезд раньше начала временного окна или опоздание после окончания временного окна и значение переменной timePenalty увеличивается на вычисленную величину.

```
if (time < population[p].car[c].stops[s].startTime)
{
     timePenalty += population[p].car[c].stops[s].startTime - time;
}
     if (time > population[p].car[c].stops[s].endTime)
{
     timePenalty += time - population[p].car[c].stops[s].endTime;
}
```

Также увеличиваем текущее время на величину `unloadingTime`, а заполненность кузова автомобиля уменьшаем на величину `needCapacity` соответствующего магазина.

```
time += population[p].car[c].stops[s].unloadingTime;
capacity -= population[p].car[c].stops[s].needCap;
```

Следующим шагом необходимо совершить проверку, не последний ли это магазин в данном цикле, если последний, то вычислить расстояние от этого магазина до хлебозавода, к которому прикреплен текущий грузовой автомобиль, а также увеличить объявленную в начале метода переменную `capacityPenalty` на величину отношения переменной `maxCapacity` к `capacity`.

```
carDistance +=
Distance(population[p].car[c].stops[s].position,
population[p].car[c].basePos);
capacityPenalty += population[p].car[c].maxCapacity /
capacity;
```

Если же текущий магазин не последний в списке запланированных, то необходимо совершить последнюю проверку, хватит ли объема в кузове чтобы полностью удовлетворить запрос следующего магазина. Если хватает, то вычисляем расстояние, пройденное данным грузовиком и текущее время следующим образом:

```
carDistance +=
Distance(population[p].car[c].stops[s].position,
population[p].car[c].stops[s + 1].position);
time += distance / 667;
```

В случае если не хватит, то расстояние к следующему магазину будет равняться сумме расстояний от текущего магазина до хлебозавода и от хлебозавода до следующего магазина. Текущее время time в свою очередь

увеличится на время этих двух поездок и еще на 30 минут, которые потратятся на загрузку кузова автомобиля. Также нужно переменной текущей загруженности `capacity` присвоить значение объема кузова текущего грузовика `maxCapacity`.

```
carDistance +=
Distance(population[p].car[c].stops[s].position,
population[p].car[c].basePos) +
Distance(population[p].car[c].basePos,
population[p].car[c].stops[s + 1].position);
time += distance / 667 + 30;
capacity = population[p].car[c].maxCapacity;
```

На этом завершаем цикл перебора запланированных магазинов и увеличиваем значение переменной `overallDistance` на значение `carDistance`. Далее завершаем цикл перебора автомобилей в плане и вычисляем искомое значение переменной `fitness`. Корректный расчет приспособленности плана крайне важен, так как от него зависит каким образом будет эволюционировать популяция.

```
population[p].fitness = overallDistance + timePenalty *
overallDistance / 60  + capacityPenalty * overallDistance / 50;
```

Приспособленность зависит от пройденного расстояния, просроченного времени, нормированного по расстоянию, и недогруженности, также нормированной по расстоянию [9]. Объем автомобиля измеряется в лотках для хлеба. В автомобиль помещается от 12 до 20 стоек, а каждой из которых задвигается по 8 лотков. Коэффициенты выбраны таким образом, что бы быть в два раза больше допустимых штрафов. Для времени было решено, что в сумме 30 просроченных минут являются приемлемыми. А насчет недогруженности решено, что приемлемы 50 выездов с полупустыми кузовами, так как этот параметр не самый важный для оптимизации.

Осталось только выйти из цикла перебора всех планов в популяции и метод вычисления приспособленности каждого плана в популяции завершен.

3.4.2 Реализация оператора скрещивания

Оператор скрещивания является основным генетическим оператором, за счет которого производится обмен генетическим материалом между особями. В случае генетического алгоритма для решения транспортной задачи производится обмен автомобилями с запланированными остановками между несколькими планами популяции. В данном случае оператор скрещивания выберет три плана и путем скрещивания их составит еще один план.

Оператор скрещивания будет представлять собой метод, принимающий переменную типа `Random` для обеспечения непрерывной генерации потока случайных величин. Сперва в методе необходимо объявить три целочисленные переменные, `i1`, `i2` и `i3`, и назначить им значения таким образом, что бы они не превышали число планов в популяции, а также не повторялись.

```
int i1 = r.Next(population.Count);
int i2 = r.Next(population.Count);
while (i2 == i1)
     i2 = r.Next(population.Count);
int i3 = r.Next(population.Count);
while (i3 == i1 || i3 == i2)
     i3 = r.Next(population.Count);
```

Далее нужно объявить список классов `Car` под названием `cross` и в него поместить все автомобили популяции `i1`, `i2` и `i3`.

```
List<Car> cross = new List<Car>();
foreach (var c in population[i1].car)
     cross.Add(c);
foreach (var c in population[i2].car)
     cross.Add(c);
foreach (var c in population[i3].car)
     cross.Add(c);
```

Также необходимо создать экземпляр класса `Plan` и в нем создать список классов `Car` следующим образом:

```
Plan plan = new Plan();
plan.car = new List<Car>();
plan.fitness = -1;
```

Теперь есть все необходимое для начала процесса скрещивания, а именно список всех автомобилей трех планов и новый экземпляр четвертого плана, который будет добавлен в популяцию в конце этого оператора. Начнем цикл `while` с условием пока список автомобилей насчитывает количество записей больше нуля. В цикле объявим целочисленную переменную `n` и присвоим ей случайное значение от нуля до количества записей в списке машин, а также сразу же добавим в новый план грузок с индексом n из списка автомобилей `cross`.

```
int n = r.Next(cross.Count);
plan.car.Add(cross[n]);
```

Затем откроем цикл, который будет перебирать все запланированные остановки для выбранного автомобиля, а в нем начнем еще один цикл, который будет перебирать все автомобили в списке `cross`, кроме того автомобиля, для которого перебираются остановки в первом цикле. Далее начнем еще один цикл, который будет перебирать все остановки автомобиля из предыдущего цикла.

```
for (int i = 0; i < cross[n].stops.Count; i++)
     for (int j = 0; j < cross.Count; j++)
     {
          if (j == n) continue;
          for (int k = 0; k < cross[j].stops.Count; k++)
               if (cross[n].stops[i].ID == cross[j].stops[k].ID)
               { … }
     }
```

В последнем цикле реализуем проверку, совпадает ли остановка `i` автомобиля `n` с остановкой `k` автомобиля `j`. Смысл заключается в том, что таким образом находятся все автомобили из списка `cross`, у которых есть тот же самый магазин, что и в выбранном автомобиле `n`. Все автомобили, где нашлись повторные магазины удаляются из списка. В случае если автомобиль `j` находился в списке `cross` раньше автомобиля `n`, то `n` необходимо уменьшить на единицу, а так же прерывать последний цикл, так как у одного автомобиля не может быть несколько идентичных запланированных магазинов для посещения.

```
cross.RemoveAt(j);
if (j < n)
n--;
break;
```

Далее закрываем все три цикла `for`, удаляем автомобиль `n` из списка `cross` и закрываем цикл `while`. Останется проверить, есть ли все магазины, запланированные для посещения за день, в автомобилях плана, и если каких либо из них нет, то добавить их любому автомобилю на любую позицию в списке остановок `stops`. Для этого создадим цикл, перебирающий все магазины из списка `shops`, в нем создадим булевскую переменную `found` со значение `false`. Далее в двух циклах переберем все магазины из списка запланированных остановок всех грузовиков и если найдем идентичные с магазинами из общего списка `shops`, то назначим переменной `found` значение `true` и прервем оба цикла следующим образом:

```
for (int c = 0; c < plan.car.Count; c++)
{
     for (int s = 0; s < plan.car[c].stops.Count; s++)
     {
          if (plan.car[c].stops[s].ID == shops[i].ID)
          {
               found = true;
               break;
          }
     }
     if (found)
          break;
}
```

После окончания этих двух циклов проверяем переменную found и если она равняется false, то это означает что в плане нет магазина из списка необходимых для посещения за день, и добавляем его к случайному грузовику на случайную позицию.

```
if (!found)
{
     int c = r.Next(plan.car.Count);
     int s = r.Next(plan.car[c].stops.Count);
     plan.car[c].stops.Insert(s, shops[i]);
}
```

Останется лишь закрыть цикл, перебирающий магазины из списка `shops`, добавить составленный новый план в популяцию и на этом оператор скрещивания разработан.

3.4.3 Реализация оператора мутации

Оператор мутации является простым в реализации и предусмотрен для выведения оптимального плана перевозок из локального минимума. Оператор мутации будет перемещать случайный магазин из запланированных остановок любого грузовика в список запланированных остановок другого случайного грузовика на случайную позицию. Метод, осуществляющий мутацию, будет называться `mutator` и так же, как и метод скрещивания, принимать переменную типа `Random`.

Метод мутации начинается с цикла, перебирающего все планы в популяции начиная с середины, так как оператор мутации должен применятся к планам, полученных путем скрещивания на текущей итерации. В цикле объявляются четыре целочисленные переменные, `c1`, `s1`, `c2` и `s2`. `s1` является индексом магазина из списка остановок автомобиля `c1`, а `s2` – индексом магазина из списка остановок грузовика `s2`. Этим переменным назначаются случайные величины следующим образом:

```
int c1 = r.Next(population[p].car.Count);
int s1 = r.Next(population[p].car[c1].stops.Count);
int c2 = r.Next(population[p].car.Count);
int s2 = r.Next(population[p].car[c2].stops.Count);
```

Далее необходимо проверить, есть ли в списке запланированных остановок для автомобиля c1 плана p магазины, и если нету, то начать следующую итерацию с помощью команды continue:

```
if (population[p].car[c1].stops.Count == 0)
     continue;
```

Если же в списке есть магазины, то нужно выбрать магазин с индексом `s1` и перенести его в список запланированных остановок автомобиля `c2` на позицию `s2` следующим образом:

```
Shop shop = population[p].car[c1].stops[s1];
population[p].car[c1].stops.RemoveAt(s1);
population[p].car[c2].stops.Insert(s2, shop);
```

Также необходимо присвоить переменной `fitness` плана `p` значение -1, так как план перевозки изменился и его необходимо пересчитать методом `FitnessFunc()`. На этом закрываем цикл перебора планов, открытый в начале метода, осуществляющий мутацию, и завершаем сам метод.

3.4.4 Метод решения транспортной задачи с помощью разработанных операторов

Первым делом нужно заполнить глобальные переменные популяции и магазинов и экземплярами соответствующих классов в соответствии с количеством работоспособных автомобилей и заказами на день. Далее нужно создать желаемое число случайных планов, где каждый магазин посещается только одним автомобилем. Рекомендуется моделировать эволюционный процесс для решения транспортной задачи с популяцией из 300 планов [6], но, тем не менее, в главе 4 исследуем зависимость качества лучшего плана от количества планов в популяции.

Теперь пришло время создать экземпляр класса `Random`. Генерация случайных чисел начинается с начального значения. При повторном использовании того же начального значения создается та же последовательность чисел. Одним из способов получения различных последовательностей является выбор зависимого от времени начального значения, что позволяет создавать различные последовательности для каждого нового экземпляра кдасса `Random`. По умолчанию в лишенном параметров конструкторе класса `Random` для генерации начального значения используются системные часы, в то время как параметризованный конструктор данного класса может принимать целочисленное значение, зависящее от количества тактов в текущем времени. Однако, вследствие конечности разрешающей способности часов, использование конструктора без параметров при создании различных объектов `Random` в быстрой последовательности приводит к

созданию генераторов случайных чисел, производящих идентичные числовые последовательности. Эту проблему можно устранить, создав единый объект `Random` вместо нескольких и передавая его в качестве параметра метода.

Далее открываем цикл `for`, который будет моделировать поколение. Число поколений равняется числу исполнения этого цикла. В этом цикле открываем еще один цикл for, который будет вызывать метод скрещивания и передавать ему ссылку на экземпляр класса `Random`. Как правило, в эволюционных алгоритмах удваивают число особей в популяции перед тем, как начать процесс селекции [8]. Поэтому текущий цикл должен исполняться столько же, сколько количество планов в первоначально популяции, так как один вызов разработанного метода скрещивания увеличивает популяцию на один план.

После того, как закончился цикл вызывающий метод скрещивания, размер популяции удвоился и нужно применить метод мутации к только что созданным новым планам. Разработанный метод мутации осуществляет мутацию всех планов начиная с середины популяции, так что его необходимо вызвать лишь несколько раз в зависимости от желаемой интенсивности мутации. В качестве параметра так же передаем ссылку на экземпляр класса Random.

После завершения процесса мутации необходимо вызвать метод вычисляющий приспособленность всех планов. Разработанный метод фитнес-функции будет рассчитывать приспособленность только тех планов, значение которых равняется «-1». Это позволит не перерасчитывать приспособленность планов, для которых приспособленность уже известна и план не менялся.

Следующим шагом необходимо сократить популяцию до первоначального размера. Для этого открываем цикл, который будет исполняться столько раз, сколько нужно удалить планов из популяции. В этом цикле будет объявляться массив целочисленных переменных с количеством элементов равным числу планов в популяции. Этот массив заполняется

значениями приспособленности соответствующего плана. Далее объявляется целочисленная переменная равная индексу максимального значения в созданном массиве и из популяции удаляется план с этим индексом.

```
for (int j = 0; j < 300; j++)
{
     int[] array = new int[600];
     for (int k = 0; k < population.Count; k++)
     {
          array[k] = population[k].fitness;
     }
     int a = Array.IndexOf(array, array.Max());
     population.RemoveAt(a);
}
```

После сокращения популяции до первоначального необходимо закрыть цикл for, моделирующий поколения, или прервать его в случае, если лучший план перевозки не изменялся в течении определенного числа поколений. Остается лишь найти в получившейся популяции план перевозки с минимальной приспособленностью и он будет решением транспортной проблемы.

4. ИССЛЕДОВАНИЕ ПОЛУЧЕННЫХ ЛОГИСТИЧЕСКИХ КАРТ МАРШРУТОВ И ВАРЬИРОВАНИЕ ПАРАМЕТРОВ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Условий транспортной задачи для теста разработанного генетического алгоритма составлялись случайным образом. Целью тестов является определение оптимальных параметров, таких как количество планов в популяции в начале и на этапе отсеивания, интенсивность мутации и оптимальное число поколений, а также попробовать визуально оценить качество рассчитанных планов.

4.1 Исследование параметром алгоритма для решения задачи с одним депо

Начать стоит с изучения вычисленных планов перевозки продукции для условия транспортной задачи с одним депо. В качестве этого депо был выбран хлебозавод номер 4 и шесть грузовиков. Емкость кузовов установлена случайной от 80 до 112 лотков. Также были выбраны 59 объектов случайным образом из списка заказчиков КУП «Минскхлебпром», которые расположены на территории города Минск. В их свойства были внесены случайным образом следующие значения: заказанный объем продукции в диапазоне от 5 до 15 лотков. Время открытия временного окна в пределах от 10 до 450 минут, считая от начала рабочего дня предприятия «Минскхлебпром», а время закрытия – позже времени открытия на величину от 30 до 60 минут. Время разгрузки автомобиля выбрано в диапазоне от 10 до 40 минут в зависимости от объема заказанной продукции.

Зависимость значения приспособленности лучшего плана от величины популяции показана на рисунке 4.1 а. На рисунке 4.1 б изображена зависимость приспособленности от числа мутаций за одно поколение, а на рисунке 4.2 показана зависимость приспособленности лучшего плана от текущего поколения популяции. В качестве значения приспособленности было взято среднее значение приспособленности десяти экспериментов.

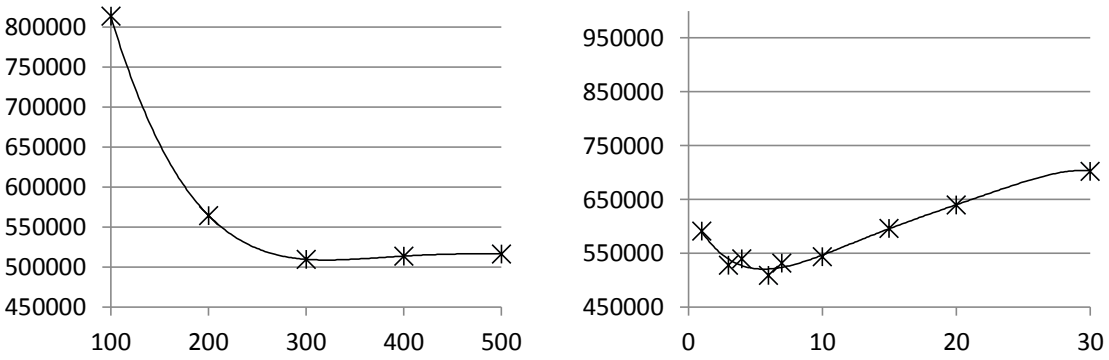

Рисунок 4.1 – а) зависимость приспособленности от размера популяции; б) зависимость приспособленности от числа мутаций.

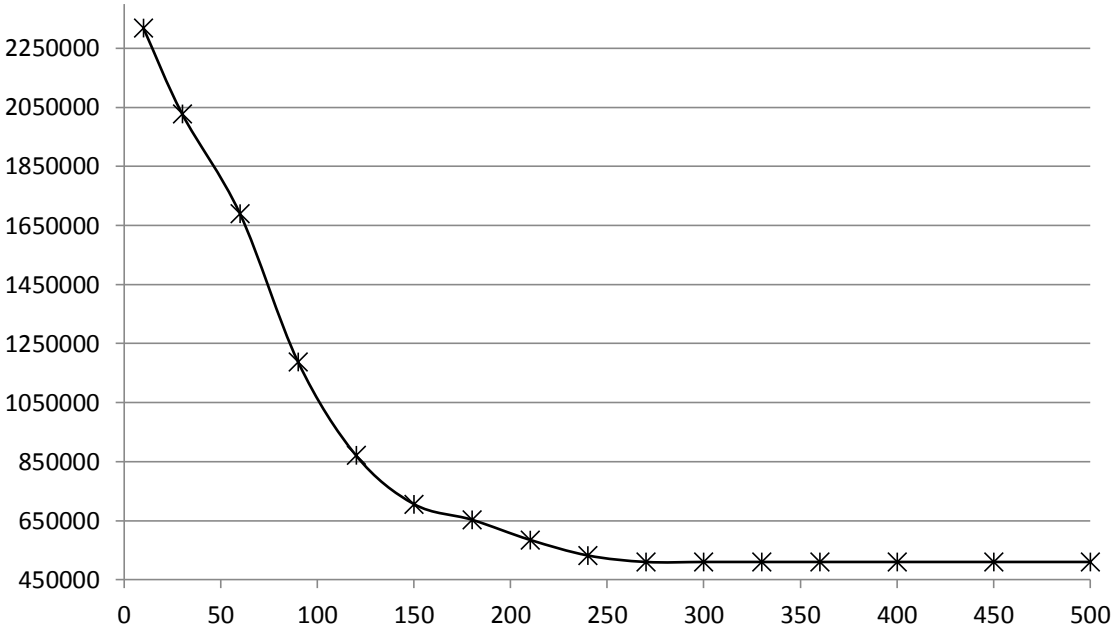

Рисунок 4.2 – Зависимость приспособленности от текущего поколения.

На рисунке 4.3 изображен один из наиболее оптимальных планов перевозки для данной задачи из одного депо и 59 объектов. Красной точкой отмечен хлебозавод №4. Синяя линия соединяет объекты, запланированные для посещения одним грузовиком. Линия изламывается в точках, где находится собственно магазин. Сумма длин маршрутов всех автомобилей изображенного плана равняется 232 километра.

Рисунок 4.3 – один из наиболее оптимальных планов для задачи с одним депо и 59 объектами.

4.2 Исследование параметром алгоритма для решения задачи с несколькими депо

Для условия транспортной задачи с несколькими депо возьмем хлебозавод «Автомат», а также номер два и шесть. В каждом из заводов будет по два автомобиля, емкости кузовов от 80 до 112. Было выбрано 56 объектов из списка заказчиков города Минск с такими же диапазонами задаваемых значений, как и в предыдущем условии.

Так же, как и в предыдущем примере, были изучены зависимости значений приспособленности от числа планов в популяции, количества мутаций за одно поколение и числа поколений и изображены данные зависимости на рисунках 4.4 а, 4.4 б и 4.5 соответственно.

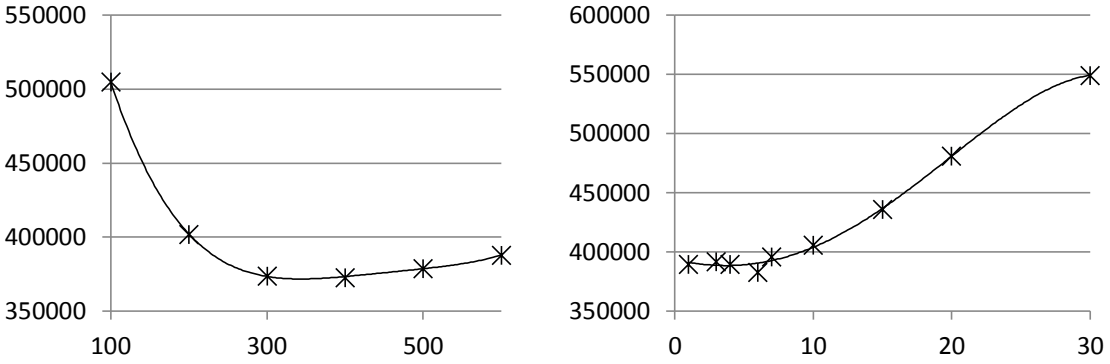

Рисунок 4.4 – а) зависимость приспособленности от размера популяции; б) зависимость приспособленности от числа мутаций.

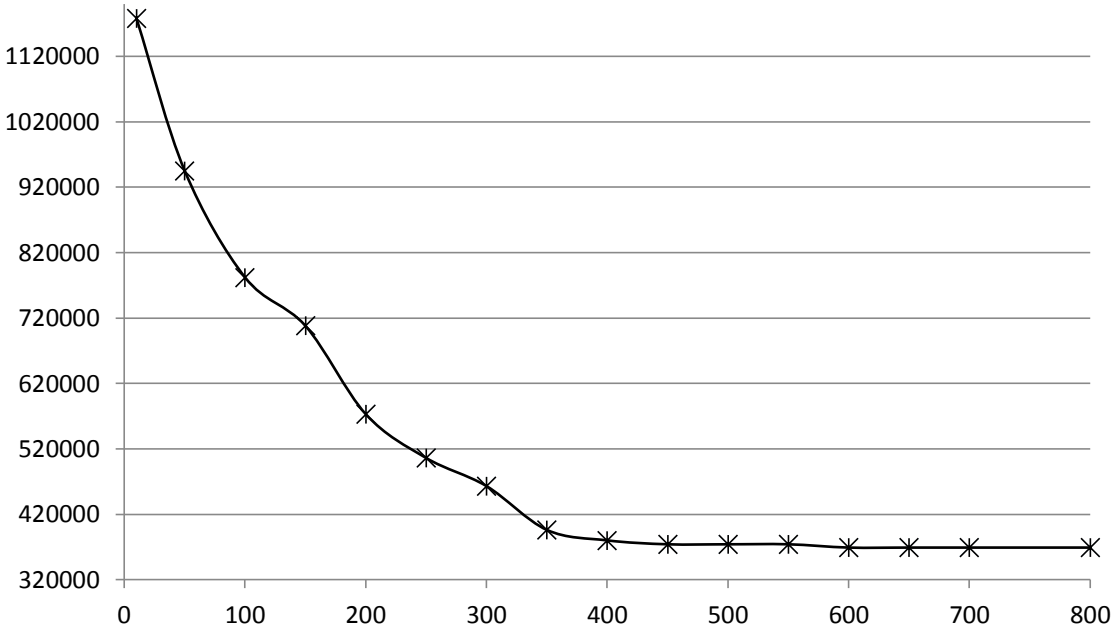

Рисунок 4.5 – Зависимость приспособленности от текущего поколения.

На рисунке 4.6 изображен один из наиболее оптимальных планов перевозки для задачи из трех хлебозаводов и 56 объектов. Так же, как и в предыдущем изображении, красными точками обозначены хлебозаводы, и синими линиями маршруты с остановками в изломах. Сумма длин маршрутов всех автомобилей изображенного плана перевозок равняется 170 километров.

Рисунок 4.6 – один из наиболее оптимальных планов для задачи с тремя хлебозаводами и 56 объектами.

Как видно по графикам, размер популяции в 300 планов является оптимальных для обоих условий транспортной задачи. Количество мутаций за одно поколение лучше всего выбрать равным шести, но разницы от одной – двух мутаций почти нет, поэтому лучше остановится на одной мутации, так как этот оператор влияет на производительность сильнее всех остальных. Что касается числа поколений необходимых для нахождения минимального плана перевозок, то минимальный план для первой задачи нашелся за 300 поколений, а ко второй – за 450.

4.3 Исследование зависимости времени получения минимальной логистической карты от количества автомобилей и заказчиков

Опытным путем было определено, что время вычисления минимального плана перевозок зависит только от числа объектов, которым нужно доставить продукцию и почти не зависит от количества транспортных средств.

График зависимости времени нахождения минимальной карты в секундах от числа автомобилей изображен на рисунке 4.7. В качестве значения

времени введено среднее время получения логистической карты пяти эксперементов. Эксперементы проводились при количестве заказчиков равным 320, что является значением приближенным к реальному. Как видим, в случае задачи с 10 автомобилями план нашелся за 337 секунд, а с 140 грузовиками – за 359 секунд.

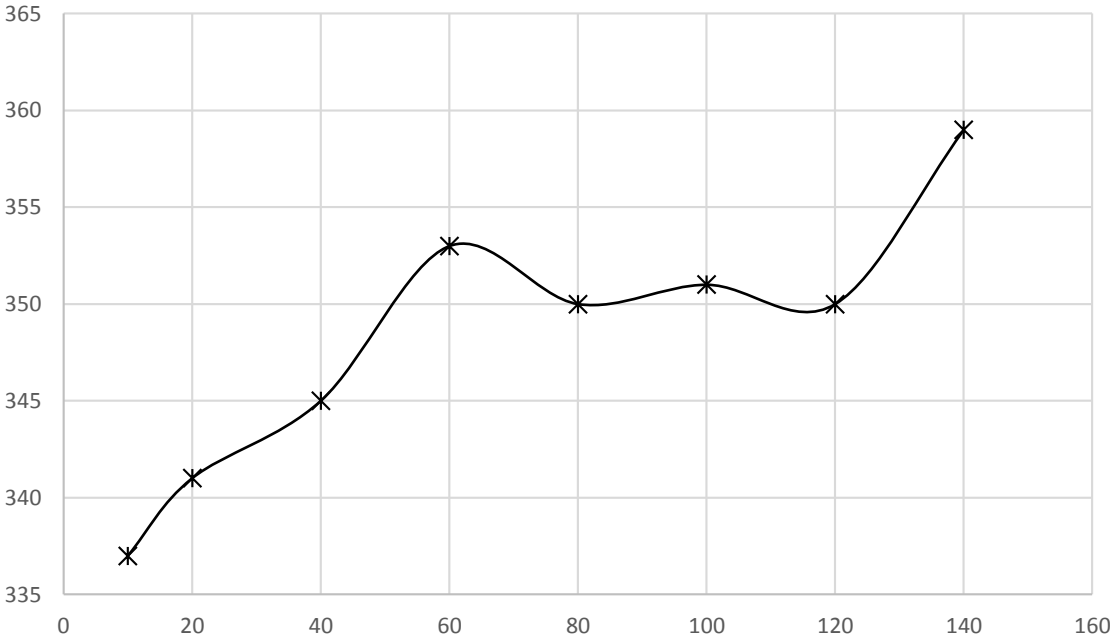

Рисунок 4.7 – зависимость времени нахождения оптимального плана от числа грузовых автомобилей.

Далее была исследована зависимость времени получения минимального плана перевозки от количества магазинов, которые необходимо посетить. В этом графике на рисунке 4.8 также внесены средние значения пяти эксперементов и число грузовиков равнялось 90.

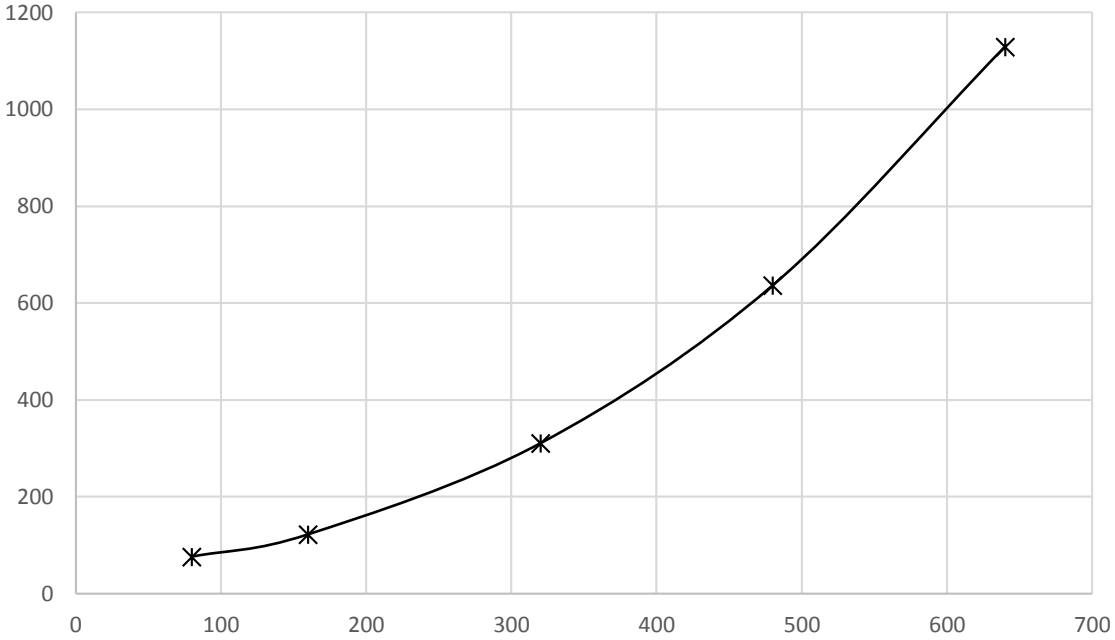

Рисунок 4.8 – зависимость времени нахождения оптимального плана от числа заказчиков, которых необходимо посетить.

На решение транспортной задачи с 80 объектами потребовалось 76 секунд, а на решение задачи с 640 объектами потребовалось 18 минут 49 секунд. Поскольку обычно заказчиков в день порядка 500, то на нахождения логистической карты маршрутов для такого условия транспортной задачи уйдет чуть больше 10 минут, что является приемлемым и вписывается в установленные рамки равные 15 минутам.

ЗАКЛЮЧЕНИЕ

В результате выполнения настоящей дипломной работы была изучена специфика работы логистического отдела промышленного предприятия на примере КУП «Минскхлебпром» и формализовано условие транспортной задачи из многочисленных требований предприятия. Полученное условие транспортной задачи включает в себя необходимость учитывать несколько депо, различный объем кузовов грузовых автомобилей, временное окно заказчика, в которое необходимо попасть с доставкой продукции, а также возможность рассчитать логистическую карту маршрутов с возвращениями некоторых грузовиков для загрузки в течение рабочего дня.

В качестве решения данной проблемы было рассмотрены несколько универсальных программных пакетов для решения транспортной задачи. Поскольку ни одно из них не удовлетворяет в полной мере составленному условию, было решено разработать собственный программный модуль для решения поставленной задачи.

Для этого было исследовано несколько алгоритмов для решения транспортной задачи и выбран алгоритм, наиболее полно отвечающий требованиям предприятия. Им оказался генетический алгоритм, который является одним из лучших для решения комбинационных задач, в том числе и транспортных с любыми условиями. При этом он быстрый и не самый сложный для реализации.

Был реализован выбранный алгоритм, описанный в главе 3, который в состоянии рассчитать приемлемую логистическую карту маршрутов, учитывая все условия поставленной транспортной задачи.

Также была исследована зависимость качества построенной карты маршрутов от некоторых параметров генетического алгоритма, таких как размер популяции, интенсивность мутаций и числа поколений, за которое минимизируется план перевозки продукции.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Система сопровождения и контроля транспорта на маршрутах [Электронный ресурс]: — 2012. — Режим доступа: http://infopark.by/sites/default/files/publication_files/minskhlebprom_sistema_soprovozhdeniya_i_kontrolya_transporta_na_marshrutah.doc. — Дата доступа: 23.05.2015.

2. Вычисление расстояния и начального азимута между двумя точками на сфере [Электронный ресурс]: — 2014. — Режим доступа: http://gis-lab.info/qa/great-circles.html. — Дата доступа: 24.05.2015.

3. Определение расстояний на поверхности Земли [Электронный ресурс]: — 2010. — Режим доступа: http://osiktakan.ru/geo_koor.htm. — Дата доступа: 24.05.2015.

4. The Google Maps Directions API [Электронный ресурс]: — 2015. — Режим доступа: https://developers.google.com/maps/documentation/directions. — Дата доступа: 23.05.2015.

5. Генетический алгоритм — наглядная реализация [Электронный ресурс]: — 2015. — Режим доступа: http://habrahabr.ru/post/254759/. — Дата доступа: 24.05.2015.

6. Решение транспортных задач с использованием комбинированного генетического алгоритма [Электронный ресурс]: — 2008. — Режим доступа: www.raai.org/conference/cai-08/files/cai-08_paper_157.doc. — Дата доступа: 24.05.2015.

7. Учебно-методическое пособие по разделу транспортная задача [Электронный ресурс]: — 2009. — Режим доступа: http://www.resolventa.ru/data/metodstud/transproblem.pdf. — Дата доступа: 25.05.2015.

8. Генетические алгоритмы [Электронный ресурс]: — 2007. — Режим доступа: http://mathmod.aspu.ru/images/File/ebooks/GAfinal.pdf. — Дата доступа: 29.05.2015.

9. Genetic Algorithm [Электронный ресурс]: — 2013. — Режим доступа: http://neo.lcc.uma.es/vrp/solution-methods/metaheuristics/genetic-algorithm/. — Дата доступа: 29.05.2015.